UCID- 20666

Code Development With The EAGLES Engineering Problem-Solving Environment

B. S. Lawver
D. W. O'Brien
M. E. Poggio
Computer Systems Research Group
Engineering Research Division

February 7, 1986

Printed in the United States of America
Available from
National Technical Information Service
U.S. Department of Commerce
5285 Port Royal Road
Springfield, VA 22161

Price Code	Page Range
A01	Microfiche
Papercopy Prices	
A02	001 – 050
A03	051 – 100
A04	101 – 200
A05	201 – 300
A06	301 – 400
A07	401 – 500
A08	501 – 600
A09	601

Introduction

EAGLES is a set of computer programs which assist in the development and use of engineering analysis and simulation codes. This paper introduces EAGLES to the developers of engineering codes. EAGLES' capabilities, functions, and tools for code development are explained. EAGLES offers the following benefits in writing and using modeling and simulation codes:

1. A consistent system of *menus* for picking functions, commands or alternative choices of any kind.

2. *Forms* to be browsed and used to enter data and other information.

3. A uniform help facility to describe events and choices while using EAGLES.

4. Easy prototyping of new codes or algorithms.

5. Modular construction of new sequences of codes and algorithms with pre- and post-process graphics display and highly interactive control of the process.

6. The ability to browse and change data and algorithms during the analysis or simulation.

Background

The EE Department develops and imports Engineering codes. The diversity of these codes and their differing human-machine interactions, graphics, data representation and report formats have frustrated engineers. Rapid prototyping of a new algorithm or application is usually not possible. These have become roadblocks to effective use of computer-aided engineering at Livermore and elsewhere.

Existing Fortran and Pascal codes typically dedicate as much as 80% of their lines to data handling, display and user interaction. However, the code developer's expertise and interest is usually the code's fundamental algorithms. Expansion and radical changes to the code's operation or data display are often not even considered. The problem then becomes how to modularize these codes, separating I/O and interfaces from algorithms; how to support a variety of codes with consistent data and human interfaces; and how to integrate all this with still other unmodified codes.

The CAE thrust area develops EAGLES *environment* to meet the problems being faced by the developers and users of engineering codes. EAGLES' goal is to improve the engineers ability to use computer-based engineering tools. There were two dominant considerations in developing the EAGLES environment:

1. The electronics engineers at Livermore use VAX computers running the VMS operating system for local interactive computing. VAXstation II workstations are coming into LLNL, supporting the variety of computer-aided engineering codes. The VAXstation offers excellent windowing, graphics, interactive techniques and a great potential for making codes more usable.

2. *Object-oriented* programming tools offer modularity and flexibility not previously available to code writers (refer to Appendix A). These tools isolate major functions of a code (e.g. algorithms, data manipulation, user interaction) and allow them to be used.

Capabilities In Place Today And Planned

EAGLES supports both multiuser VAX computers and the newer VAXstation workstations. Today EAGLES is available on the Tektronix 4105/7 and RetroGraphics terminals, as well as the VAXstation II. In the future, EAGLES may be moved to the UNIX operating system. This would allow it to use more specialized hardware like the IRIS workstation which supports high speed 3D graphics.

The EAGLES Development Project is currently mid-way through its initial phase, using engineering controls tools as a test-bed. Controls codes and algorithms under EAGLES include:

1. The *M* interpretive language for interactive matrix calculation, manipulation and the building of control systems supporting *matrix*, *system* and *signal* data types.

2. A spreadsheet-like matrix editor uniquely suited to handling matrices of double precision complex numbers.

3. A graphical block diagram editor for hierarchical design of feedback control systems supporting Multi-input/Multi-output (MIMO) systems development and modification.

4. Numerical libraries including *Linpack* and *Eispack*.

5. Controls-specific graphics display of data.

6. An interactive engineering controls simulator (planned for EAGLES integration next year).

Most of the functions and capabilities described in this paper are already available to controls engineers. EAGLES plans to continue development and support of the EAGLES environment and to add more engineering tools. Engineering modeling and circuits codes, and simulation with animated graphics are prime candidates for EAGLES.

Functions Available with EAGLES

EAGLES offers considerable advantages for prototyping new capabilities. Tools for the code developer include *panels* and *wrappers* to aid in data display and interfaces, as well as user *menus* and *forms*. Menus and forms are simple program functions a code developer can use to retrieve the user's menu choice or the input data from a form. Control of applications and graphical display of results, at this point, are more application specific.

Panels and Windows

The ability to run EAGLES with a dynamic, windowed display improves the use of engineering codes. Panel and window tools are crucial to the development of the EAGLES menu and form systems as well as human-computer interfaces for engineering codes within EAGLES.

Professor Kenneth Joy of UC Davis developed the concept of panels to interface programs to a variety of workstation and terminal windowed display systems. Panels are virtual windows. The abstract interface isolates the low level workstation or terminal characteristics from the code.

EAGLES' panels, though well suited to run on workstations, are also supported on ASCII-based graphics terminals. On an ASCII-based terminal EAGLES runs one process at a time. But various panels that contain menus, forms, and other tools may be displayed on the screen (overlapping if necessary or desired). The bit-mapped graphics display on the VAXstation II makes more flexible use of windows. Several codes may be running in different windows, each of them using more windows to display menus, forms, and tools.

Menus

EAGLES provides menus as a consistent user interface for selecting alternative functions, commands, algorithms or data. EAGLES menus are horizontal and dynamic pull-down type menus (a la Apple's Macintosh). Terminal cursor keys or the VAXstation's mouse are used as a pointing device for making menu selections.

Forms

Forms are used to interactively enter data, parametric and program control information. The engineer responds to the codes query by browsing a captioned form and filling in the necessary information. Default values may be pre-entered on the form under control of the code. Some entries may be mandatory, others optional.

Help

EAGLES offers a help facility that allows help information to be displayed on the screen in a help panel. The help text is kept in a file external to the application program and may be edited by any text editor available on the computer system running EAGLES.

Wrappers

EAGLES uses wrappers to integrate existing codes and code libraries. The concept of a wrapper was formalized by research done independently by Professor Joy of UC Davis and Dr. Brad Cox at Schlumberger-Doll Research. EAGLES refined ideas from both models to form our definition of wrappers. The wrapper converts data to-and-from EAGLES and the format used by a Fortran or Pascal engineering code.

Wrappers handle the data conversion, range checking, error reporting and sequencing through a set of functions. The summary below may help in the understanding of EAGLES wrappers, their intent and implementation.

What is an EAGLES wrapper?

- a mechanism to combine several dissimilar codes, libraries and subroutines to make a new capability.
- a database interface between codes, libraries and subroutines.
- a control sequencer of several codes to make them appear as one.
- a mechanism to use codes in the EAGLES object-oriented environment with other tools in any desired way.

An EAGLES wrapper can be implemented as

- an *object* of the object-oriented programming environment (an Objective-C program).
- a command file controller.
- a menu command sequencer.
- Fortran, Pascal or C programs.
- a database management system.

Graphics

EAGLES uses graphics to effectively display data from engineering codes. EAGLES attempts to isolate graphics from the code, supporting wrapped application-specific data display for better flexibility and interactivity.

EAGLES supports both the DIGLIB and GKS low level graphics libraries. DIGLIB is important for the many existing engineering codes that were developed using it. GKS is important because most new workstations support this standard.

Integrating Engineering Codes Into EAGLES

Integrating a new code or capability into EAGLES is usually a cooperative effort between the code developer and the EAGLES development team. There are three ways new capabilities may be integrated into EAGLES:

1. Incorporate into a generated menu pre-existing codes which will run with their own data handling and user interface when selected from the menu. This is the easiest way, integrated into EAGLES in minutes. But this does not take advantage of EAGLES wrappers or forms, or the interactivity and flexibility of the object-oriented environment. This can be recommended as a first step for a code developer to get an introduction to EAGLES.

2. Separate an existing code (or develop a new code) into algorithms, display, data handling and program control. Computational algorithms (code libraries and subroutines) may then be wrapped. Menus for picking various algorithms or functions may be created. Forms for entering data, and other information may be created. Application-specific graphics may be developed. This offers many of the benefits of EAGLES, yet requires minimal contribution by the EAGLES development team (see figure 1).

3. Develop an interactive controller using EAGLES development tools (Objective-C, LEX and YACC) to manipulate wrapped algorithms and data objects (see figure 2). The engineering controls language *M* was developed in this fashion and offers the most flexibility.

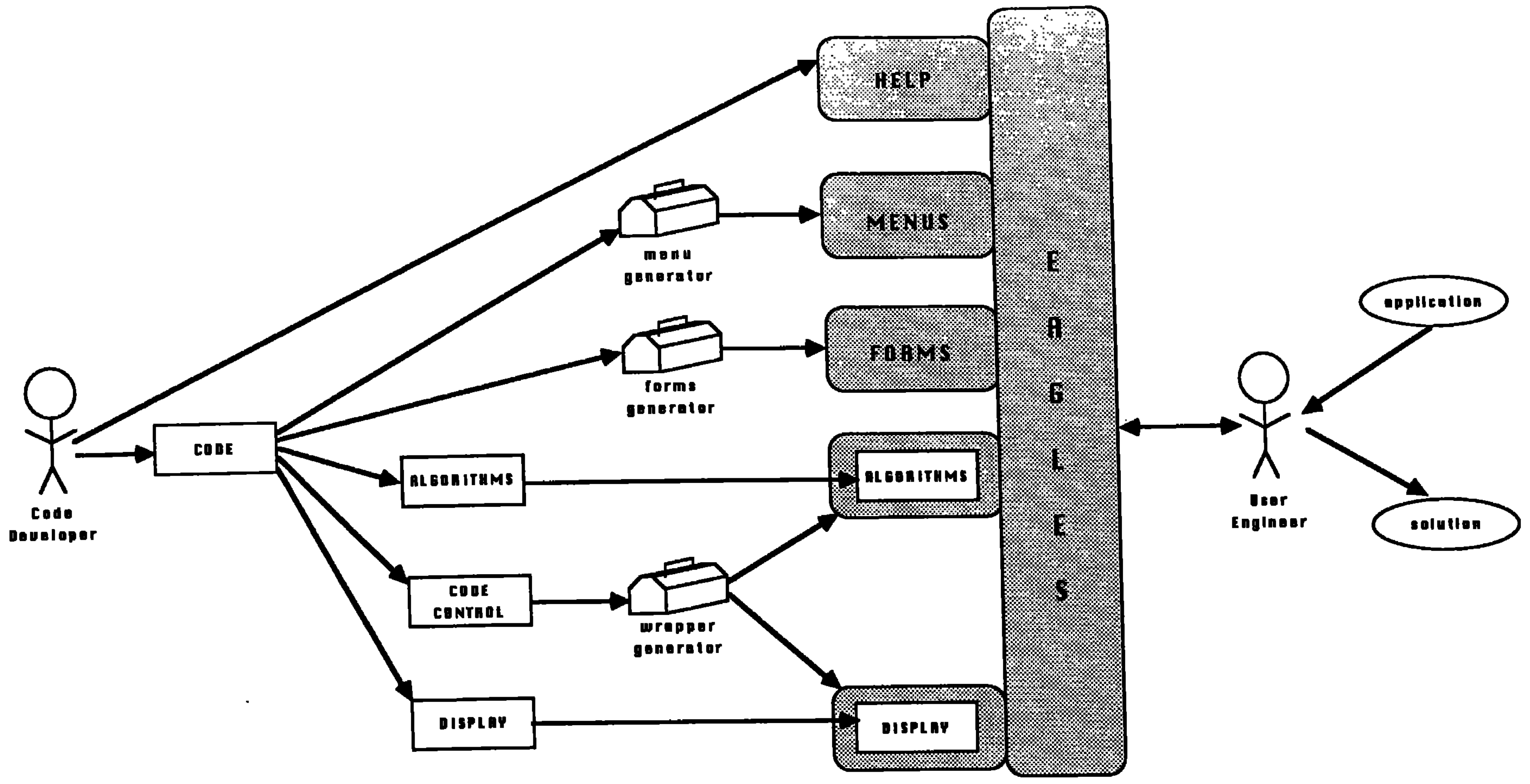

Figure 1. A pictorial representation showing the EAGLES/code intermediate level of integration. EAGLES development team members interface, if necessary, with the code developer to assist in using the tools provided by EAGLES (represented here by toolboxes).

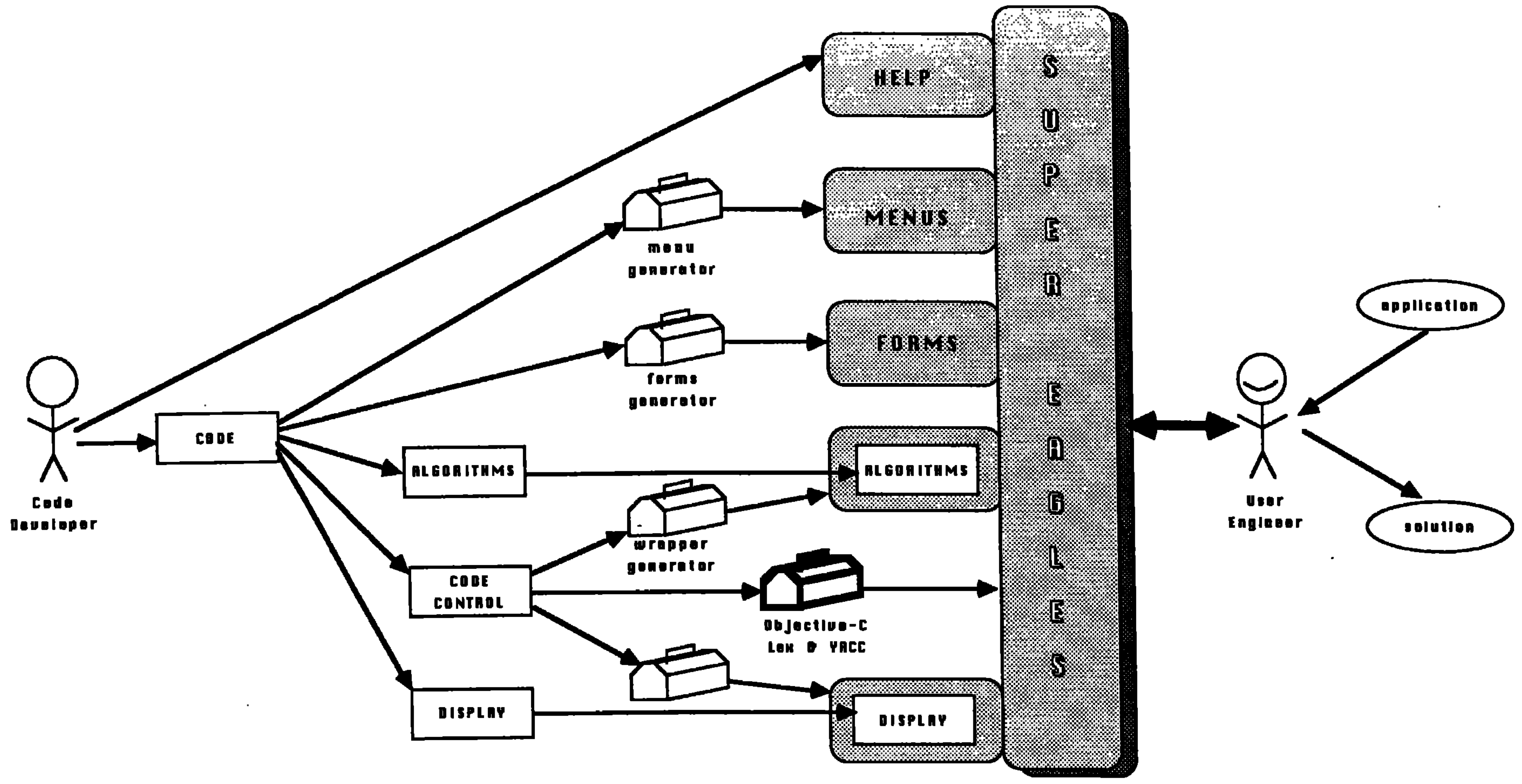

Figure 2. A pictorial representation showing the EAGLES/code high level of integration. This type of integration requires that the code developer invest significant time in learning EAGLES tools and interfacing with EAGLES development team members.

New controls capabilities and applications are easy to develop in M. And it is possible to interactively try alternate designs, or modify a design or data while in the process of running a simulation.

Code developers or engineers wishing to bring a new modeling or simulation capability under EAGLES need to consider the following:

- The code developer should have access to the Fortran or Pascal source to an existing code or code library. It will probably have to be modified or broken apart.

- The fundamental algorithms, their use, and limitations must be well understood by the code developer.

- The code developer will be expected to specify and develop the data structures and sequence of operations to be used with the code.

- The code developer must commit to share the responsibility and invest the time to integrate the code into EAGLES.

Once the interfaces are well understood, the code developer can construct an EAGLES environment with wrappers, menus and forms. EAGLES has tools for menu and form construction and controllers for their use by the engineering code. A tool is also available for wrapping code libraries, subroutines and unmodified codes. With this set of tools it is easy for the code developer to prototype a user interface of menus and forms and later revise the user interface without making a sizable investment in the interface development cycle.

Menu Generator Tool

EAGLES has a *menu generator* that allows code developers to build a menu structure that is consistent with other previously generated EAGLES menus. Integration of menus into a code is done through a menu controller which ensures that the manipulation of menus is consistent among various EAGLES applications. The menu controller can display a hierarchical set of menus to execute other tools as well.

Form Generator Tool

EAGLES has a *form generator* which allows a code developer to define captioned forms to be filled in by the engineer as the code is being used. The form generator is picked from the EAGLES menu and uses a form to aid in the design of new forms. Incorporation of a form into a code is done through a carefully controlled interface similar to the menu controller.

Wrapper Generator Tool

A *wrapper generator* is available to the code developer to assist in integrating codes or libraries into the EAGLES environment. Today the wrapper generator is limited to Fortran numeric data structures. Automatic wrapper generation is still under development and more complex data structures will be available. The wrapper generator can be picked from the EAGLES menu and uses a form to be filled in describing data structures.

Appendix A

Object-oriented programming

Object-oriented programming technology has developed over the last fifteen years. Until recently it was only available on very high cost, experimental workstations. Object-oriented programming is now available in Xerox's SMALLTALK80, Symbolic's FLAVOR, and PPI's Objective-C. Object-oriented programming has provided fresh, innovative approaches to system design, the most recognizable being Apple's Macintosh. Object-oriented languages employ several concepts not previously found in programming languages.

An *object* can be thought of as a data structure with predefined behaviors. *Classes* couple this data with a structure of *methods* (functions or algorithms) providing powerful building blocks that are well suited for implementing interfaces (i.e. menus, forms and wrappers). The ability to "bind" data and algorithms at the time the engineer runs an application offers the flexibility to browse through the input, algorithms and results and to make changes to each, perhaps choosing a different algorithm or method in the middle of a problem's solution.

A hybrid object-oriented language, Objective-C, was selected for developing the EAGLES environment. This language adds object-oriented constructs to the C programming language. These constructs implement the important object-oriented programming concepts of object/class structure, *encapsulation*, and *inheritance* while offering full access to the C programming language for more computationally intensive chores. The features of Objective-C lend themselves well to the goals of the EAGLES environment.

Objects
: virtual machines that require and retain some data themselves and do work. Objects are dynamically bound at run-time. One may think of objects as data records which carry the descriptions of the functions that may be carried out on them, exhibiting a behavior (e.g. a matrix of numbers becoming the transpose of itself upon receiving a message to do so).

Messages
: requests made of an object to exhibit one of its predefined behaviors (functions).

Classes
: the collections of related methods. Classes of methods are built in a hierarchy or tree structure. This structure allows a method to share data and behaviors with all other methods above it in the tree. This is referred to as inheritance. Every object associated with a class can respond to any message that invokes a method in that class's inheritance tree. Classes are also the factories that produce new objects of a similar type and contain the shared part of these similar objects.

Methods
: related methods (behaviors or functions) are grouped in classes. A method is invoked by sending an object a message.

Encapsulation
: the powerful by-product of the object/class structure which keeps a clean, simple program interface to arbitrarily complex objects.